DEEP REINFORCEMENT LEARNING ON 1-LAYER CIRCUIT ROUTING PROBLEM

BY

YIHAO ZHANG

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Adviser:

Professor Martin D F Wong

Abstract

In VLSI design, routing is the step that determines the paths for circuit nets and interconnections. While routing can be a very complex process involving time, congestion and space information, the problem can be modelled as a maze routing problem. In specific, given a 2d array and a set of start nodes and end nodes, the agent is trying to optimize the solution by connectivity and path length. Traditionally, the routing problem is solved using graph search techniques such as Lee's algorithm. The result produced by graph search algorithms relies heavily on the order of routing. While some simple heuristics are available, the result is not stable because simple heuristics take greedy approaches and neglect the long-term reward. The recent development of deep learning, especially deep reinforcement learning, can be a good approach to finding better ordering on attacking the routing problem. We introduce a reinforcement learning approach to the traditional 2-point nets in 1-layer maze routing problem.

Table of Contents

CHAPTER 1: Literature Review

Routing is the problem of how to connect macro blocks on chips such as ASICs and SoCs. Routing is difficult in the sense that the scale of the problem is huge, the paths are not self-revealing, and the geometry can be complex. There are several types of routing problems: 2-point nets in 1-layer, multi-point nets and multi-layer routing. In this study we focus on the 2-point nets in 1-layer problem. This is the most basic kind of routing problem. Its history can be traced back to shortest path problems in networking planning [1] [2]. The problem is later found useful in electronic wiring [3]. The problem described in VLSI design is called maze routing. Maze routing is a classic method to model 2-point nets in 1-layer problem.

Maze routing assumes a gridded geometry, meaning each grid is a square and the maze is made of grids. In gridded geometry, pins and wires both lie in cells. Grid layout implies that the wires can go in vertical or horizontal directions. It also assumes wires of the same width. Pins lying in grids suggests that pins have no or uniform physical size. Moreover, the blockage is also represented in uniform sizes.

Maze routing can be solved using either serialized or parallelized methods. The serialized method wires the connection in a specific order. Parallelized connection will extend wires simultaneously. In this study we focus on serialized connection.

A major breakthrough in this problem is the Lee router [3] [4]. The Lee router uses the idea of serialized connection. It finds the shortest path and uses backtracking to establish the connection. The path will be marked as blocked after successful connection, and the process continues. Lee router uses breadth-first search technique [5]. Breadth-first search is time consuming and has to start a new problem for each connection to be made.

In recent years, machine learning has seen huge improvements because of hardware upgrade. Neural network architectures, reinforcement learning theory and computer vision applications have seen great development. AlphaGo beating human player marks a new milestone in reinforcement learning [6]. Supervised learning such as ImageNet and natural language processing has also been a hot topic.

In AlphaGo, the board is in a gridded geometry just as the maze routing problem. The researchers use reinforcement learning and Monte Carlo tree search to optimize the decisions. We think that it is possible to create a router that learns from experience without any human correction or supervision.

CHAPTER 2: Learning a Static Maze through Deep Q Learning

Overview

Maze routing requires the agent to route the maze with the best possible connectivity. If the agent completes the routing by picking one pair of start node and end node at a time, the order will have an impact on result connectivity, as in figure 1. Every path created will influence the current 2d array because every connection will create new blockage. In the situation in figure 1, it is not possible to use any simple heuristics to derive the optimal order. Instead of using any feature related heuristics, we propose using a deep learning approach to transfer this problem from a heuristic design to a learning problem.

Various sources online support the conclusion that deep learning is able to learn solving a maze problem (with one pair of start and end nodes) with an end-to-end solution. Compared with

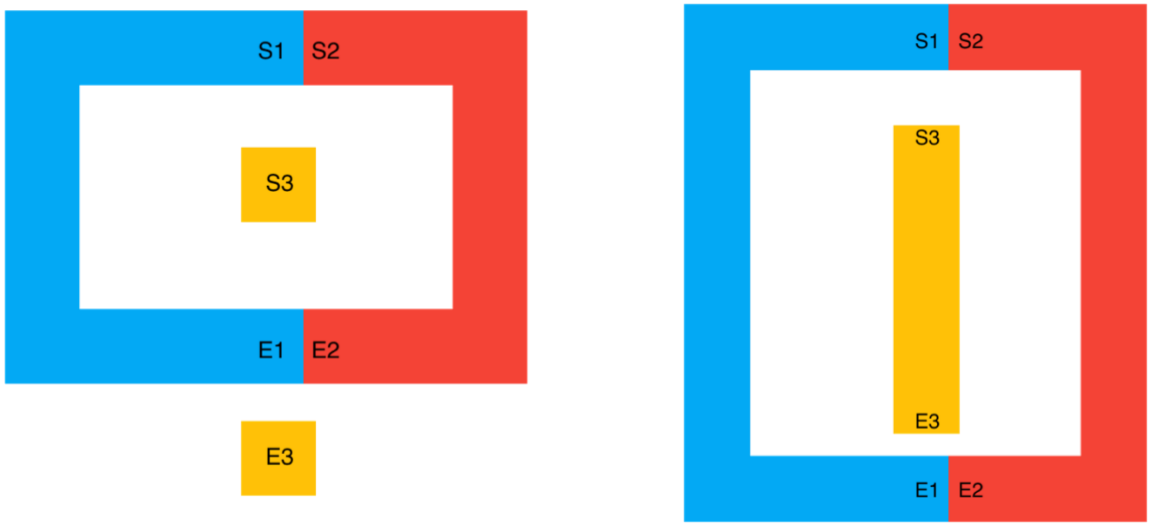

Figure 1: Result of poor ordering (left) and good ordering (right)

simple single-node-pair maze, our environment has more variations and the environment interacts more with the decision. Despite the fact that the training uses multiple different start and stop points, the maze blockage is not changed, hence the agent will only need to learn the

decision rewards at every point. Maze routing has more possible states. The route of the current pair is affected by the blockage created by all the previous pairs. Besides, every blockage created by the current node can be vastly different depending on different configurations. The route (or dynamically created blockage) can either create a line across the two points or take a detour to achieve the goal. Given the condition that it is not possible to try all the possible orderings, we assume that deep learning can increase the routability by learning a small fraction of the sequences.

Model and assumptions

We assume a fixed-size maze with fixed start and end nodes. The goal is to find good enough connections in a small amount of exploration. We assume that the route will be connected by shortest route algorithms. In my experiment, I used A* search for shortest path between start and end nodes [7]. The path should mostly fall in between the bounding box containing start and end nodes [8]. Though we can assume that the bounding box can contain the path, the routes are not self-revealing.

In circuit design, blockages are not only created by wires, they can also be pre-determined as existing module and routes. Here we ignore this setting, because we only want to examine the potential to use deep learning to learn routing with dynamic blockages. The blockage created by every route is dynamic and unpredictable, so we reduced the setting of fixed blockage to reduce the model complexity. Another reason is that blockages are treated the same in the deep learning framework. Since we are not separating fixed blockage and dynamic blockages, replacing potential fixed blockage with dynamic blockage is acceptable.

Because every module can have several pins in and pins out, the wires will usually follow an L shape pattern. In this experiment, we will just use random pin placement for simplifying the model.

The first step in this experiment is to determine if routing a pre-determined 2d array is possible. The agent will learn previous experiences and save the weights of all decision sequences. Over time, the agent will be able to achieve a connectivity better than random and some simple heuristics. Since it is learning the order of connection, we limit the information to previous connection and possible future connections. This arrangement is similar to partial observability applied in multiple deep learning experiments in video games [9].

As a result, we come up with a deep learning definition of this multiple source maze game. A chain of connection will lead to completion of the maze. The agent will only know the previous action and the next possible action. As reinforcement learning is designed to mimic how an animal or human reacts to positive or negative rewards, a successful connection will be rewarded, and, intuitively, a failed connection will be negatively rewarded.

We first tried Q learning to better understand the reinforcement learning technique we are about to use. Our goal is to use deep Q learning, which is an extension of traditional Q learning with neural net. A Q learning model can be used in solving a maze problem with single start and end with ease. The algorithm will remember the past routes and the reward associated with past actions. Then it will optimize based on experience.

Deep Q learning (DQN in the rest of this thesis) is famous for its application to playing Atari games. Deep Q learning combines Q learning with deep neural network. DQN uses similar Q function to map state and action to a potential reward. Instead of Q table, it uses Q neural network. The property of gradient descent update and backpropagation of the neural network

enables learning more complex models and anticipating future rewards more accurately. Figure 2 below contains the detail reinforcement learning algorithm that we implemented.

Algorithm and specific settings

Algorithm 1 Deep Q Learning

procedure DQN
  Initialize replay memory D with capacity N
  Initialize value function Q with random weights θ
  Initialize total episode count M, total step count T
  while $episode < M$ **do**
    while $step < T$ and game is not over **do**
      With probability ϵ select random action a
      Otherwise select $a_t = argmax_a(Q(\theta(s_t), a; \theta)$
      Simulate action a_t, receives reward r_t and new board x_{t+1}
      Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess θ_{t+1}
      Store transition in D
      Sample random mini batch $(s_j, a_j, r_j, s_j + 1)$from D
      Set $y_i = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta)$
      Perform gradient descent step on loss function

Figure 2: DQN algorithm

Reinforcement learning usually consists of several components including agent, state, action etc. Following are the full definitions:

Agent: The agent acts based on anticipated reward. It will choose the action with highest anticipated reward to perform.

State: The state for stationary 2d array is encoded as the previous connected pairs, the reward from the previous connection, and the next connectable pairs.

Action: The action is to connect one of the remaining pairs. The action is not to choose one pair, but only connect one pair. The choosing will be the max pooling from all the possible actions.

Reward: For this experiment, we tried two types of rewarding system. The first is a length-based rewarding. Since any connection created will be adding blockage, which can

potentially block the others, a shorter length will imply fewer future blockages. The second type of reward is purely connection based. If the connection is successful, the reward will be positive; otherwise, the reward will be negative.

Discount factor: Discount factor is used to help the model converge. It can also balance the exploration and exploitation ratio. A discount rate is applied to the potential reward calculation in the training stage. In the following formula for anticipated cumulative reward in equation 1, γ is the discount factor, r is the reward for the current chosen action, action is a, and state is s. In our setting the discount factor will be set to 0.9.

$$\mathbb{E}[r+\gamma \max_{a'} Q(s',a';\theta_{-i})||s,a]$$

Equation 1: Equation for cumulative result

Environment: The environment is a maze with width and height equal to r. In this experiment r ranges from 60 to 1000.

Policy: The policy here is to find the best order to connect the pins.

Q-value: Q-value is the anticipated cumulative reward generated by the neural net.

Replay: Replay is an important part of reinforcement learning. The intention is to let the agent learn strategy from experience. The strategy we took is to let the agent learn a batch of the experience every time, the same method DeepMind uses for Atari games [10]. The motivation behind this is to eliminate the strong correlation between the steps. Also, learning from random batches can increase the chance of exploration because replaying random batches can prevent the optimizer from being trapped in a local minimum [11]. The batch size will be set to 32.

Reward function: Reward is used to provide feedback for both each intermediate step and the final completion reward. In traditional goal terminating games, the last step will have higher rewards. In our settings, completion is not the ultimate goal. The goal is to achieve higher

connectivity. As a result, we set reward to be 1 for successful connection, and -1 for unsuccessful connection.

We use random connection to compare against DQN. Using random connection as a benchmark, the main focus of this experiment is to see if a DQN agent can beat random connection. The connection problem is not similar to the classification problem, as random guess can lead to a decent score. The focus is on whether a reinforcement learning agent can find a better sequence to make more connections from all the random or greedy learning processes.

We used two other simple heuristics as our greedy learning algorithm, one is shortest route first and the other is longest route first. Shortest route first has the advantage of picking up the local connections first and connecting the longer ones later. Longest route first heuristics has in mind that the longer routes are more likely to fail at the later stage, so ensuring the longer routes first might have a positive impact on the overall connectivity. There is no concrete evidence that any static strategy will result in a better connectivity.

Training DQN agent for fixed maze

As a theoretical foundation, we want to confirm that a reinforcement learning agent is able to learn a fixed maze. In this experiment we tried several training settings, including various 2d array sizes, node pairs, learning rates, learning episodes and discount factors.

The training of this part is done on a local machine. The tensorflow is working on CPU rather than GPU, since the iteration number does not have to be very large to reach convergence for a fixed maze. The result from DQN agent is compared against the result obtained by random agent and two simple heuristics. Figure 3 is the result of routing. Yellow paths are connected routes, and blue dots are pins that cannot be connected.

Figure 3: Fixed 2-d array example result

Results

Table 1 lists the quantitative result for 200*200 sized 2d array with 25 pairs of pins, and 100*100 sized 2d array with 25 pairs of pins. For 200*200 2d array, after 1000 rounds of training, the DQN router scores higher connectivity than random router. The total length is similar.

Table 1: Quantitative result on a fixed 2d array

	DQN	RANDOM
Mean Connectivity	0.816	0.712
Reward	14.8	8.6
Total Length	3373	3339

For 100*100 sized 2d array, we used 3 algorithms to compare against DQN router. DQN again scores better than the rest on most occasions. Due to the testing cases assignment, the overall connectivity is fluctuating in figure 4. However, the reinforcement learning router

managed to stay at the top. Figures 4 and 5 show the connectivity over training iterations for each heuristic. Figure 6 is a bar chart describing the connectivity. As we can see, DQN can be a very robust model in learning fixed 2d maze without any supervision.

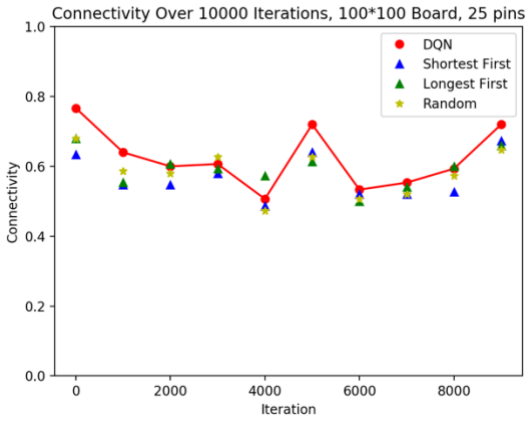

Figure 4: Fixed 2d array percentage difference

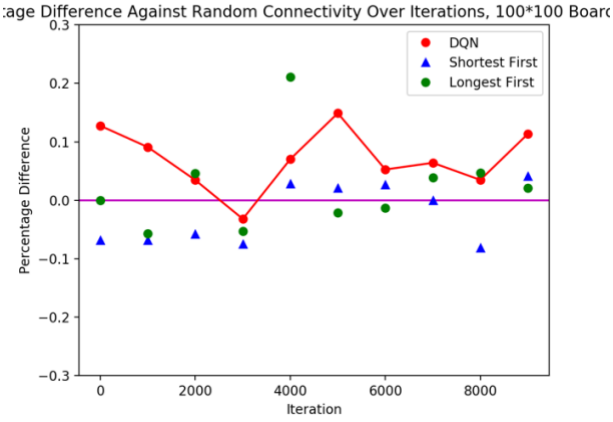

Figure 5: Connectivity over 10000 training iterations

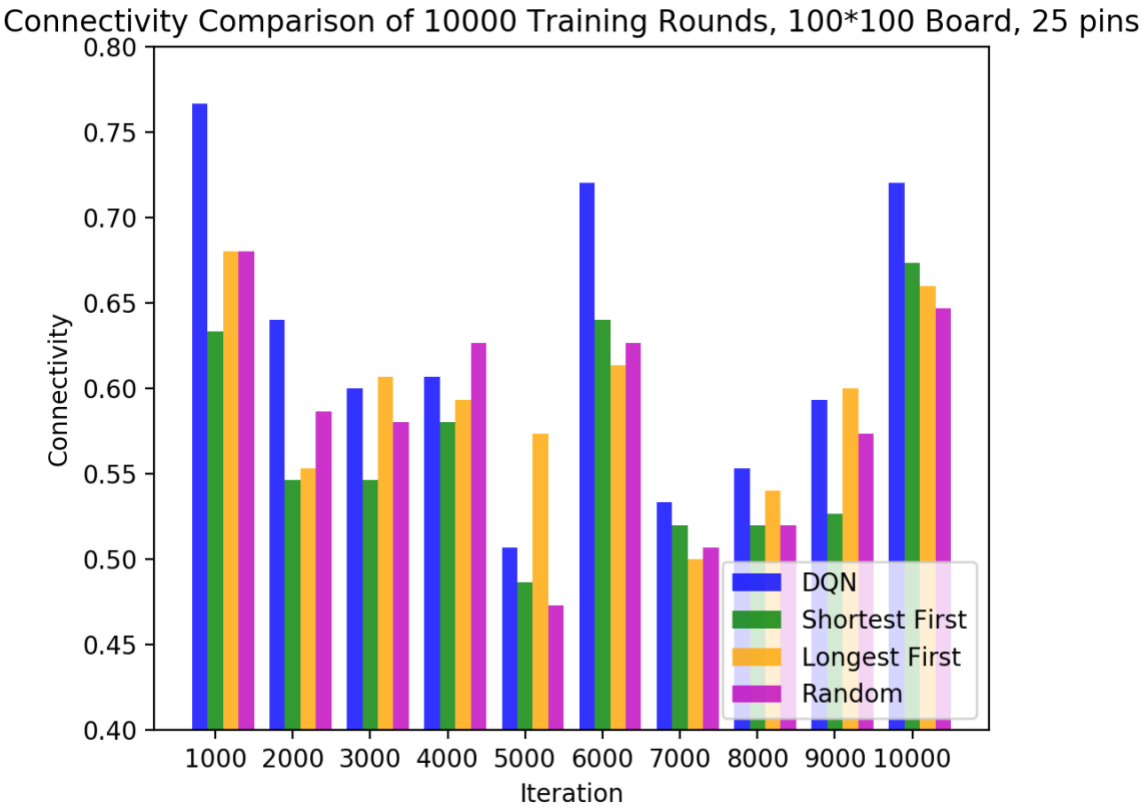

Figure 6: Bar chart for connectivity

CHAPTER 3: Learning General Maze Routing through Deep Q Learning

Since we have confirmed that DQN is able to learn a static maze, the next step will be to determine if transforming the learning result to an unseen set of mazes is possible. Learning strategy from discrete examples can be challenging. The position information can be less useful. The context of the whole 2d array must be taken into consideration. An implicit challenge is to infer possible future blockage.

In traditional circuit design, usually the route will be in a bounding box that surrounds the start and end points. The assumption is that most of the route will not be outside of the bounding box. In addition, using A* algorithm for single route connection can guarantee the shortest connection.

Increase the awareness of environment

To increase the agent's 'awareness' of the maze 2d array, it is essential to add more information to the network. With only start point and end point information, the connectivity gain is marginal. After 2000 rounds of training, the connectivity is still fluctuating with no sign of stabilizing.

During the experiments we found that a better connection solution usually results in shorter average length. For example, on a fixed 2d array, the average path length on the best router is about 10% shorter than that of a random router. As a result, we think that the average path length might have some implicit impact on connection. As we are developing a reinforcement learning model, a reward function can influence the learning rate and learning result [12]. Previously, we fixed the reward to 1 and -1 for successful and failed connections. As a result, the solver learns purely from connectivity result. The experiment we did on reward function is to add a reward that is linearly dependent on the actual connection path length. The

reward also has to reflect the connectivity. As a result, we set the reward for successful connection to be 0.5 + width/pathLen. We set the reward for failed connection -1. In this construction, both connectivity and path length will be considered.

Since the 2d array information has to be encoded into our reinforcement learning model, we considered a few options. First, we thought about CNN model. The reason is that CNN model is widely used, and it is very popular for all sorts of image-related reinforcement learning with few total actions, because of the Atari Game application and OpenAI gym environment. While having that in mind, we also noticed that this experiment is different from other simple image related games such as Atari or Pinball. The action space is much larger and more dynamic. The information is not exactly related to the shape of the object. The decision is more detailed and has less room for adjustment because the route will be connected once the pairs are chosen.

Connected and potential connections

We represent the potential connectivity as congestion level in the bounding box. As discussed previously, most of the paths lie within the bounding box containing the start point and the end point. Assuming that the possibility of any cell within a bounding box is part of the path is uniformly distributed, all cells in the bounding box will have the same weight. One cell can exist in multiple bounding boxes. The value in each cell represents the potential congestion.

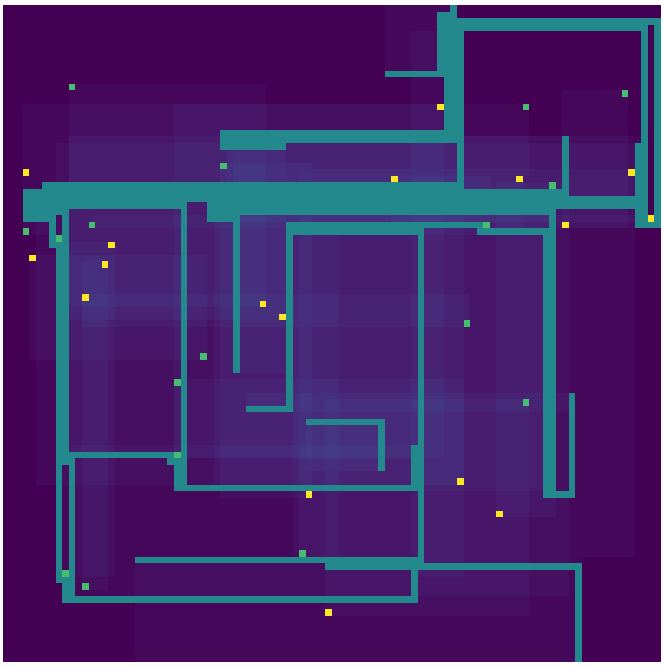

Figure 7: Example result with congestion level

In figure 7, the shadows represent potential congestion level. The teal lines are the paths between connected start points and end points. The outstanding pins are left alone. It is obvious from figure 7 that most of the connections are formed within the shadowed area, confirming our previous assumption that paths are formed within the bounding box. In addition, more paths lie in the area where the congestion level is higher. Note that randomized pins can cause lower connectivity because there are higher chances for paths to form boxes and cut lines that completely block the potential, while modular design has the advantage of easier, more formulated pin connections.

New parameter setting for transferability test

Reinforcement learning parameters (only the different ones will be addressed):

State: the current 2d array with congestion level

Environment: 60*60, 300*300, 600*600 and 1000*1000 2d array

Reward: 0.5 + width/pathLen or -1if a connection cannot be made

Results

*Table 2: Quantitative result for 60*60 2d array with 30 pairs*

60*60, 30 Pairs	DQN	Shortest First	Longest First	Random
Mean Connectivity	0.55	0.46	0.51	0.47
Mean Length	56.92	57.62	68.21	60.39

*Table 3: Quantitative result on 300*300 2d array with 30 pairs*

300*300, 30 Pairs	DQN	Shortest First	Longest First	Random
Mean Connectivity	0.73	0.68	0.70	0.69
Mean Length	258.84	306.10	279.26	280.91

*Table 4: Quantitative result of 600*600 2d array with 40 pairs*

600*600, 40 Pairs	DQN	Shortest First	Longest First	Random
Mean Connectivity	0.6	0.55	0.5	0.52
Mean Length	587.48	584.11	628.63	594.14

*Table 5: Quantitative result for 1000*1000 2d array with 40 pairs*

1000*1000, 40 Pairs	DQN	Shortest First	Longest First	Random
Mean Connectivity	0.64	0.60	0.59	0.54
Mean Length	926.78	919.18	924.34	925.86

Tables 2 to 5 show the quantitative result for each configuration after sufficient amount of training. Figures 8 to 11 show the result for each testing round, including connectivity, path

lengths and mean connectivity. From this experiment, we conclude that reinforcement learning can be used on the pin connection problem. As after a large number of training rounds, DQN connector significantly outperforms random and simple heuristics. The three heuristics used are random, shortest path first and longest path first. In the results these three heuristics do not have hierarchical difference. For smaller 2d array, the number of iterations is larger because the total time needed for training is determined by iteration number and the time for each iteration. A smaller 2d array will significantly reduce the cost of A* algorithm. On larger 2d array, completing the connection once takes much longer.

The path length is also calculated during each experiment. In 60*60 and 300*300 2d arrays, it is obvious that shorter average length will result in higher connectivity. A difference more than 10% in average path length is common between the best connection result and the poorest connection result. On a larger 2d array, for example 1000*1000 and 600*600, the average length is a poor indicator as the best connector has the third best average length. A possible reason could be that the best router connects more pins, and those additional pins take a longer than average distance.

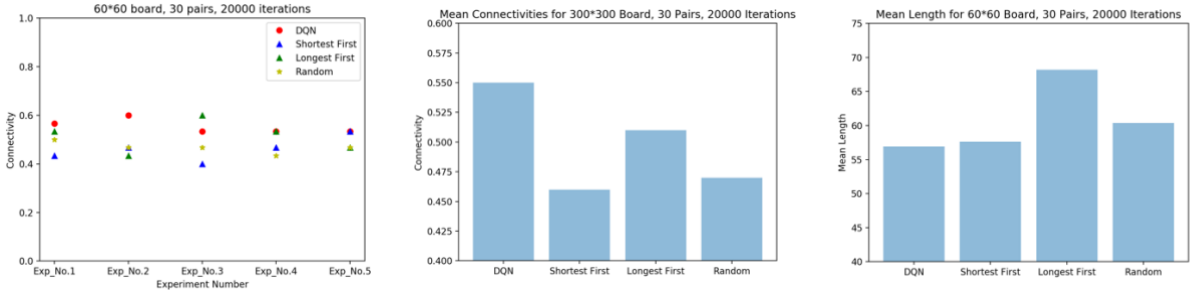

*Figure 8: 60*60 array, with 30 pairs, 20000 training iterations*

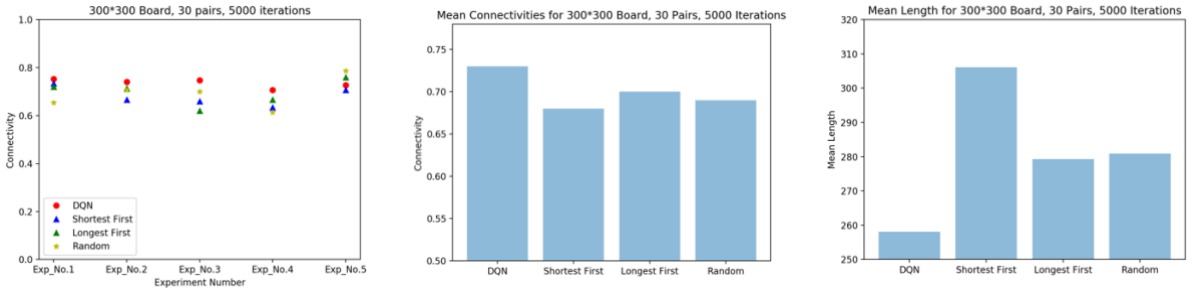

*Figure 9: 300*300 array, with 30 pairs, after 5000 iterations*

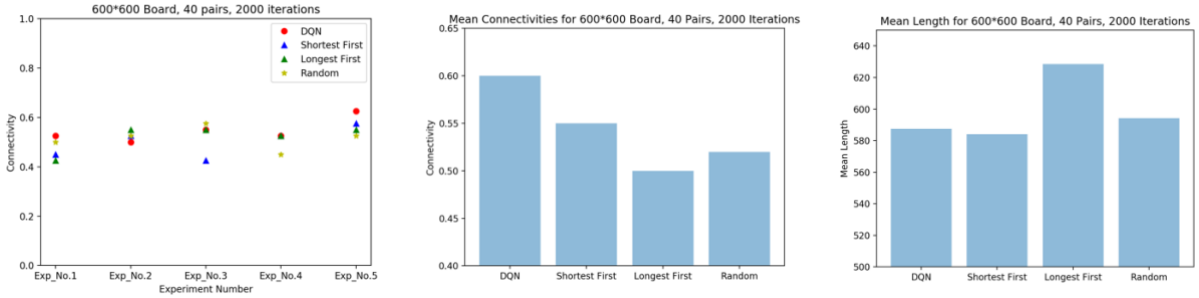

*Figure 10: 600*600 array, with 40 pairs, after 2000 iterations*

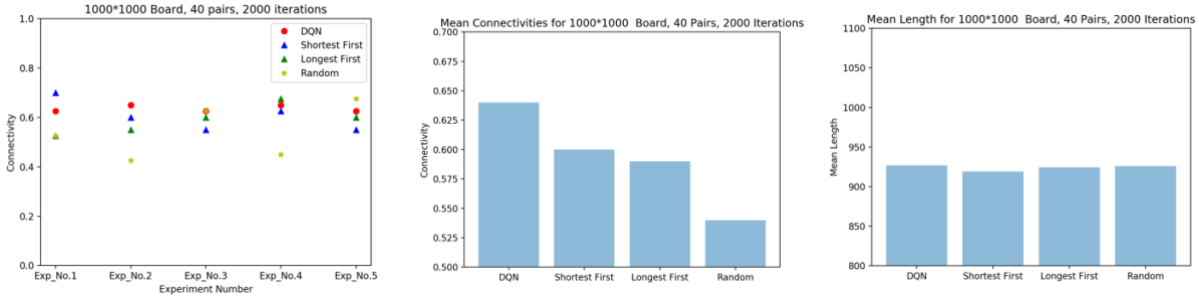

*Figure 11: 1000*1000 array, with 40 pairs, after 2000 iterations*

Figure 12 shows connectivity over iterations and percentage difference compared against random connection. The graph is generated for a 300*300 array size and 30 pairs. The random order router and the other two routers have fluctuating performances over the test period. The DQN router can stay on the top after 2000 iterations and even improve itself.

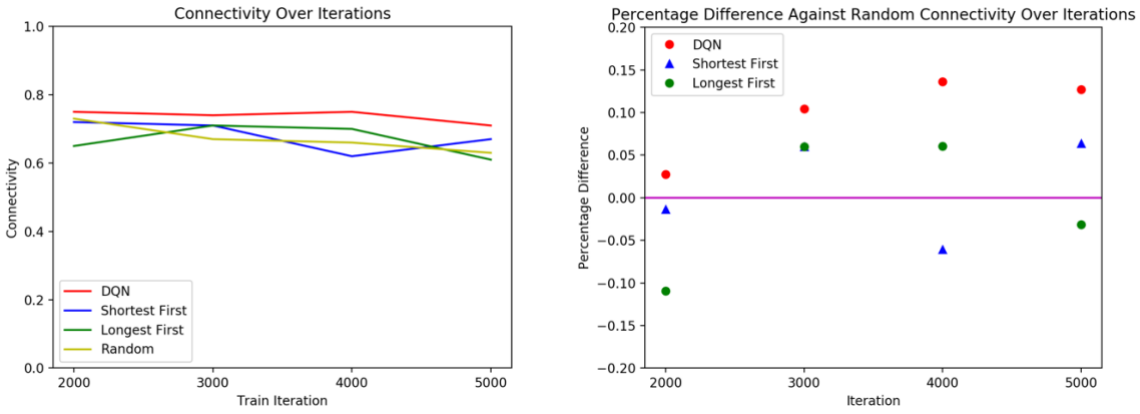

Figure 12: Connectivity over training iterations and percentage differences, using random as reference

Chapter 4: Model Transferability on Different Numbers of Pins

The previous experiment steps are to examine the possibility of having a router that connects stable configurations and unseen configurations based on learned experience. The next stage of experiment is to attempt training with different numbers of pins, while 2d array size remains unchanged.

The impact of expanding the array size is substantial when the matrix is small with a poor random connection. When size is limited, more critical blockage will be formed. When the 2d array size is big enough to accommodate all the connections, the order of connection starts to matter as we can find that there is little difference in connectivity for 40 pairs of nodes in a 300*300 2d array and in a 1000*1000 2d array.

In this experiment we use a 300*300 2d array size. The reason is that path finding algorithms take extra time when the 2d array size is bigger. For example, the A* algorithm we are using will have an exponential increase in time consumption. Moreover, the connectivity does not increase meaningfully with the 2d array size when the 2d array size is large enough. Randomly arranged pins will be very likely to spread all over the 2d array and the path will be extended with the 2d array size. By looking at the results above, we can confirm that the average path length is almost equal to the width or height of the 2d array. When the path length is positively associated with the width, limiting the size of the 2d array makes practical sense.

In this experiment, we use the same setting as the previous experiment on discount rate, 2d array settings, state and actions. The pins are randomly generated, but by different amounts. The training process is expected to take longer, so we used 20000 total training iterations. For each iteration, with its own 2d array settings, the learning process will take place in small batches.

Training process

The training is done on an AWS machine with 1 industrial GPU. The training process, either done on a cloud machine or a dedicated local machine, can be discontinued and restarted at any time, because all the intermediate files can be saved as either json or h5 files. The intermediate result saves the neural network settings, models and weights at the moment. When the training process restarts, the program will load these settings. This will ensure that the training can continue for long-term purposes.

The training process is divided into 20 rounds, each containing 1000 iterations of training. Each training includes 20 episodes of mazes of size ranging from 30 to 45 and each episode contains batched replay. Intermediate testing is done for every 1000 iterations. For every test round, we use 20 randomly generated testing 2d arrays and apply our four routers. The test result is then averaged and compared.

Results

After 20000 iterations of training, I ran a test with 1000 randomly generated maze routing configurations. Table 6 contains the connectivity result after training. Overall, DQN agent has better connectivity, but by only a small margin.

Table 6: Quantitative result after 20000 training iterations

300*300, 30 Pairs	DQN	Shortest First	Longest First	Random
Mean Connectivity	0.61	0.58	0.53	0.56

After 8000 iterations, the DQN router more often scores higher connectivity than the other three. After 12000 iterations, the model is converging, and the fluctuation is smaller. As we can see from figure 13, the average connectivity from the DQN router will stay at the top. And the percentage difference against random is more stabilized after 12000 iterations. The best

performing heuristic is shortest first. It outperforms DQN in 5 test rounds, 4 of them in the first 8000 iterations. The worst performing heuristic is longest first; it usually stays under even random, showing the possibility that when connecting the longest route first, the amount of blockage can be huge.

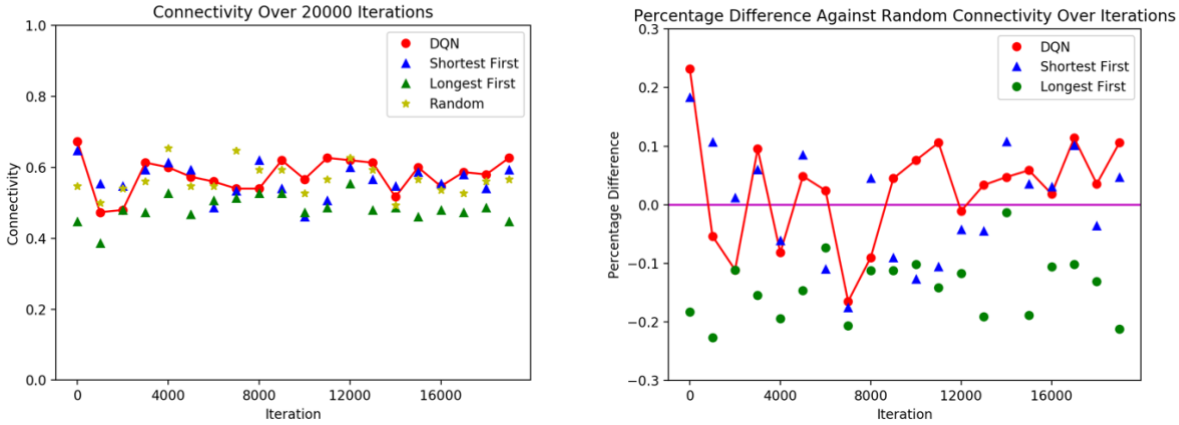

Figure 13: Connectivity over 20000 training iterations, and percentage differences against random

Figure 14 is a comparison of connectivity for the last 10 test rounds (the last 10000 training iterations). It is more obvious than the graph comparison of every test round, because the last 10 testing rounds show more stability and that reinforcement learning router can outperform the rest. It is interesting to see that random selection outperforms DQN for once. Closer scrutiny reveals that this is because some random connection has exceptionally high connectivity. The outliers dramatically increase the average connectivity.

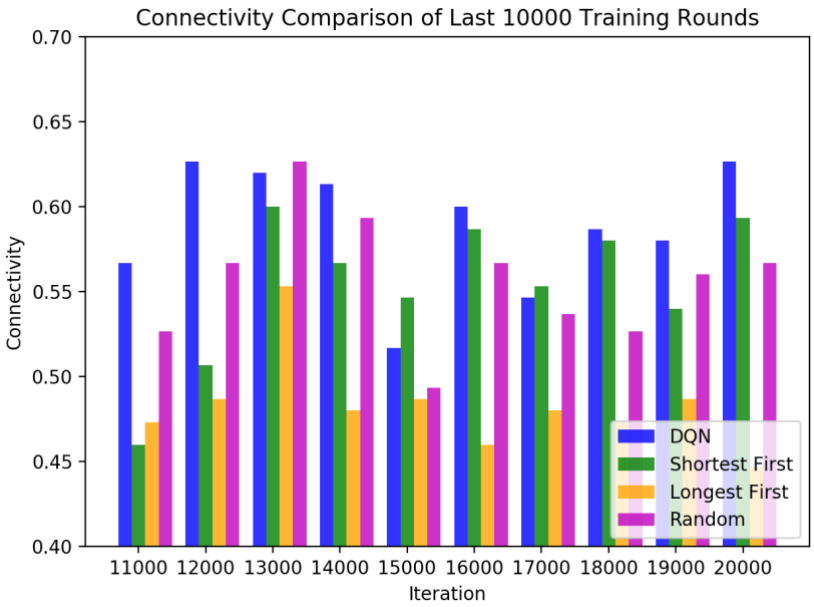

Figure 14: Comparison of the last 10000 iterations

Chapter 5: Conclusion, Performance Analysis and Future Work

Overall the performance of DQN is better than random and simple heuristics. We do observe difficulty in training when there are many variables, namely in stage 3 of the experiment.

The bottleneck of the training process lies in the path finding algorithms. A shortest path algorithm is inevitable in maze routing, and A* is a popular choice even for computer games [13]. A shortest path algorithm will take $O(|E|+|V|log|V|)$, where E is the total edge count and V is the total vertex count. The time consumption will scale considerably when the size of the maze increases. A possible way to work around it is distributed computing, where training can be split into different work machines and somehow weaved back together.

Another technical difficulty is the scalability in maze array sizes and more realistic pin arrangement. As the training and testing is on randomly generated examples, supervised learning method might improve the result in a shorter period of time. However, supervised learning requires many real examples. Bringing in more realistic training examples requires a lot of collective effort; take ImageNet for example [14]. A possible way around that is to have a high-quality router generate examples and let the neural net model learn from the generated samples.

Corner case handling is also a problem with a generalized reinforcement learning model. Sometimes in the testing process, we observe outliers, having either very high or very low connectivity. A more fine-grained optimization is needed for the outliers.

References

[1] G.B. Dantzig, "On the Shortest Route Through a Network," *Management Science,* vol. 6, pp. 187-190, 1960.

[2] E. Moore, "The shortest path through a maze," *International Symposium On the Theory of Switching Proceedings,* pp. 285-292, 1959.

[3] C. Lee, "An algorithm for path connections and its applications," *Electronic Computers,* pp. 346-365, 1961.

[4] D. Hightower, "The Lee router revisited," *ICCAD,* pp. 136-139, 1983.

[5] F. Rubin, "The Lee Path Connection Algorithm," *IEEE Transactions on Computers,* vol. C-23, no. 9, pp. 907-914, 1974.

[6] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Driessche, T. Graepel and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature,* vol. 550, p. 354–359, 2017.

[7] N.H. Barnouti, S.S.M. Al-Dabbagh, and Naser, "Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm," *Journal of Computer and Communications,* vol. 4, pp. 10-25, 2016.

[8] O. Faroe, "Local search for final placement in VLSI design," *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281),* 2001.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," in *NIPS Deep Learning Workshop 2013*, 2013.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M.A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, and D. Ku, "Human-level control through deep reinforcement learning," *Nature,* vol. 518, pp. 519-533, 2015.

[11] S. Lange, T. Gabel, and M. Riedmiller, "Batch Reinforcement Learning," in *Adaptation, Learning, and Optimization*, Springer, Berlin, Heidelberg, 2012, pp. 45-73.

[12] L. Matignon, G. Laurent, and N.L. Fort-Piat, "Reward function and initial values: Better choices for accelerated Goal-directed reinforcement learning," in *Lecture Notes in Computer Science*, Springer, 2006, pp. 840-849.

[13] X. Cui, H. Shi, "A*-based Pathfinding in Modern Computer Games," *IJCSNS International Journal of Computer Science and Network Security,* vol. 11, pp. 125-130, 2011.

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and F. Li, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV),* vol. 115, pp. 211-252, 2015.